主要内容

- 访问控制与授权
  - 基本概念和方法
  - 基于角色的访问控制
- 访问控制机制的应用实例
  - UNIX文件访问控制模型
  - OAuth2.0

三、网络安全基础设施

- 防火墙的基本原理（linux的防火墙）

访问控制(Access Control)

访问控制包括两部分：

- 认证Authentication: 你是否是你声称的那个人?
  - 确定是否允许访问（access）
  - 认证：人→机器；机器→人；机器→机器

> **访问控制机制**：实现依据安全策略对使用系统资源进行控制，且仅许可授权实体（用户、程序、进程或其他系统）依据该策略使用系统资源。 --RFC4949

- 授权 Authorization: 你能做什么?
  - 对行为进行限制
  - 施加什么限制?
- 注意：访问控制和授权有时被当作同义词

授权

- 信息安全的核心
- 对已经获得认证的用户行为进行限制
- 和身份认证的关系
  - 身份认证：二值化（通过/拒绝）
  - 授权： 更细粒度
- 经典方法
  - 访问控制列表 Access Control Lists (ACLs)
  - 访问能力列表 Capabilities (C-lists)

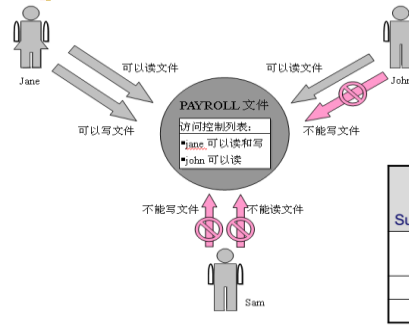
访问控制策略

- 自主访问控制（Discretionary Access Control– DAC）
  - 基于请求者身份和访问规则（授权）控制访问，规定访问者可以（或不可以）做什么。
  - 传统方法
- 强制访问控制（Mandatory Access Control– MAC）
  - 通过比较具有安全许可（表明系统实体有资格访问某种资源）的安全标记（表明系统资源的敏感或关键程度）来控制访问。
  - 军事信息安全需求
- 基于角色的访问控制（Role-based Access Control– RBAC）
  - 基于用户在系统中所具有的角色及相关访问权限规则来控制访问
- 基于属性的访问控制（Attribute-based Access Control- ABAC）
  - 基于用户、被访问资源及当前环境条件来控制访问

自主访问控制

- 自主访问控制（Discretionary Access Control,DAC）
  - 由客体的属主对自己的客体进行管理，由属主自己决定是否将自己客体的访问权或部分访问权授予其他主体。
    - 客体：资源（文件、目录、消息、邮件、程序等）
    - 主体：用户、应用、进程等
- 在自主访问控制下，一个用户可以自主选择哪些用户可以共享他的文件
  - 例如，Linux系统中的两种自主访问控制策略
    - 9位权限码（User-Group-Other）
    - 访问控制列表ACL（Access Control List）

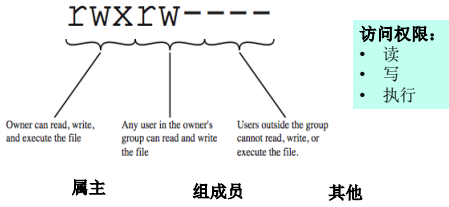
自主访问控制示例 1



Object / Subject	PAYROLL
Jane	Read Write
John	Read
Sam	

自主访问控制示例 2

- Unix文件的访问控制



访问权限：
- 读
- 写
- 执行

Lampson 的访问控制矩阵

- 场景：不同用户访问资源（操作系统或数据库中）
  - 主体 **Subjects** (用户)：对应一行
  - 客体 **Objects** (资源)：对应一列

	OS	记账程序	财务数据	保险数据	工资数据
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
记账程序	rx	rx	rw	rw	rw

访问控制矩阵

- **访问控制矩阵拥有全部相关信息**
- 例如
  - 1000个用户和1000个资源
  - 矩阵表项： 1,000,000
- 如何管理庞大的矩阵?
  - 稀疏矩阵，占用存储空间大
- 在访问任何资源之前，都需要检查该矩阵
- 如何提高性能?
  - 分解为多个可管理的片段（行，列）

访问控制列表Access Control Lists (ACLs)

- ACL: 按列存储访问控制矩阵
- 例子：蓝色的列

	OS	记账程序	财务数据	保险数据	工资数据
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
记账程序	rx	rx	rw	rw	rw

访问控制列表（续）

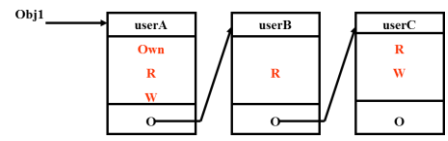
- 上表中 保险数据Insurance data 对应的访问控制列表

  (Bob， -), (Alice, rw), (Sam, rw), (记账程序, rw)

- 说明
  - 只要一个客体被访问，就引用相对应的访问控制矩阵的列，查询是否允许该访问操作

访问控制列表的结构实例

- 每一个访问控制列表（ACL-Access Control List）是客体（目标对象）的属性表
  - 适合确定某个资源中各主体的访问权
  - 但在查询特定用户的所有权限时，效率低下



访问能力列表 (C-Lists)

◆按行存储访问控制矩阵

◆例子：红色的行

	OS	记账程序	财务数据	保险数据	工资数据
Bob	rx	rx	r	---	---
Alice	rx	rx	r	rw	rw
Sam	rwx	rwx	r	rw	rw
记账程序	rx	rx	rw	rw	rw

访问能力列表（续）

◆上表中 Alice对应的访问能力列表

(OS，rx), (记账程序, rx), (财务数据, r), (保险数据, rw), (工资数据， rw)

◆说明

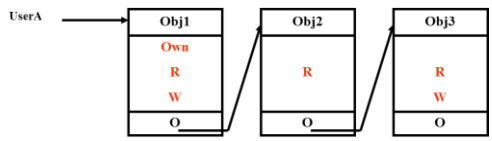
❖只要一个主体需要执行某个操作，就会引用相对应的访问控制矩阵的行，查询是否允许该访问操作。

访问能力列表的结构实例

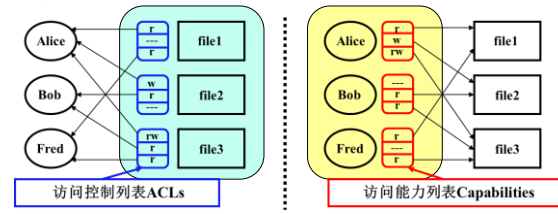
◆确定对某特定资源具有访问权的用户

❖安全控制上的优势：容易确定一个特定用户所拥有的访问权集合，实现授权机制

❖难以确定对特定资源具有指定访问权的用户列表（与ACL相反）



对比：ACLs vs Capabilities



◆ 左图： 注意箭头方向

◆ ACLs：需要建立用户和文件之间的关联

对比：ACLs vs Capabilities

- 访问控制列表ACLs
  - 管理自己的文件
  - 面向数据的保护
  - 容易改变一个特定资源的权限
- 访问能力列表 Capabilities
  - 容易实现授权机制
  - 更难以实现
  - 学术界的研究

混淆代理人问题（Confused Deputy）

一个典型的安全问题

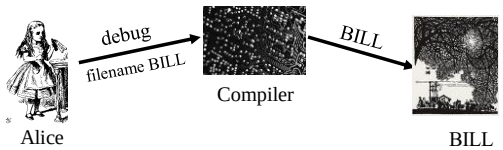
- 两个资源
  - 编译器Compiler和 BILL文件 （财务信息）
- 访问权限
  - Compiler可以“write”文件 BILL
  - Alice 能够调用 Compiler，并指定 debug输出的文件名
  - Alice 不能“write”文件 BILL

访问控制矩阵

	Compiler	BILL
Alice	x	---
Compiler	rx	rw

x: 执行
r: 读
w: 写

混淆代理人 问题（续）



- Compiler 可以作为Alice的代理（**deputy**）
- Compiler 发生混淆（**confused**）
  - Alice不允许写文件 BILL
- Compiler混淆了自己的权限和Alice的权限

混淆代理人 问题

- 采用 访问能力列表 容易实现授权机制
- 例如：
  - Alice将自己的访问能力列表授权给编译程序 complier
- 防止混淆代理人问题
  - 建立授权authority和意图purpose之间的关联
  - 建立逐级授权机制

混淆代理人 攻击实例

◆例1：Cross-Site Request Forgery (CSRF,跨站请求伪造)

- ❖攻击者在网页中植入恶意代码或链接
- ❖当受害人的浏览器执行恶意代码或者点击链接后，攻击者就可以访问那些被害人身份验证后的网络应用。

◆例2：通过CSRF攻击获取家用路由器远程管理权限

- ❖攻击者获取网络日志，确认内网IP
- ❖通过修改路由器设置，使得通过外网可以直接访问到内网IP，并开启路由器的远程管理

基于角色的访问控制： RBAC模型

◆基本概念

- ❖用户（User）：访问系统的个体，具有与之关联的用户ID
- ❖角色（Role）：组织内部进行命名的工作职能
- ❖许可（Permission）：访问权限，对特定访问模式的认可
- ❖会话（Session）：用户与其被分配的角色集合的激活子集的映射

◆RBAC（Role-Base Access Control）

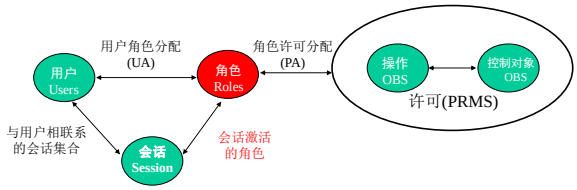
- ❖根据需要定义各种各样的角色，并设置合适的访问权限，而**用户根据其责任和权利被指派为不同的角色**，并由此取得该角色所对应的所有权限。
- ❖这样整个访问控制过程就被分成了两部分，即**访问权限与角色相关联**，**角色再与用户相关联**，从而实现了用户与访问权限的逻辑分离

基于角色的访问控制： RBAC模型

“A role can be defined as a set of actions and responsibilities associated with a particular working activity”

◆ 角色与组的区别

- ❖组： 用户集
- ❖角色：用户集＋权限集

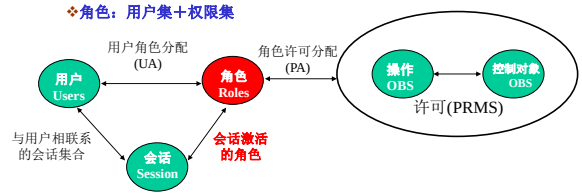


基于角色的访问控制： RBAC模型

RBAC认为权限授权实际上是Who、What、How的问题
在RBAC模型中，who、what、how构成了访问权限三元组，也就是“Who对What（Which）进行How的操作”

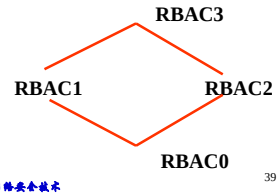
◆角色与组的区别

- ❖组： 用户集
- ❖角色：用户集＋权限集



基于角色的访问控制模型

- Sandhu等人提出的RBAC96/ARBAC97/ARBAC02模型族
  - 基本模型 RBAC0：最小功能
  - 角色分层模型 RBAC1：增加角色层次，继承关系
  - 角色约束模型 RBAC2：增加约束
  - 统一模型 RBAC3：包括上述所有功能



RBAC的三个安全原则

- 最小权限原则（即细粒度控制原则）
  - 分配给与某用户对应的角色的权限不超过该用户完成其任务的需要
- 责任分离原则
  - 在完成敏感任务过程中分配在责任上互相约束的两个角色
- 数据抽象原则
  - 通过权限的抽象来体现，如财务操作用借款、存款等抽象权限。

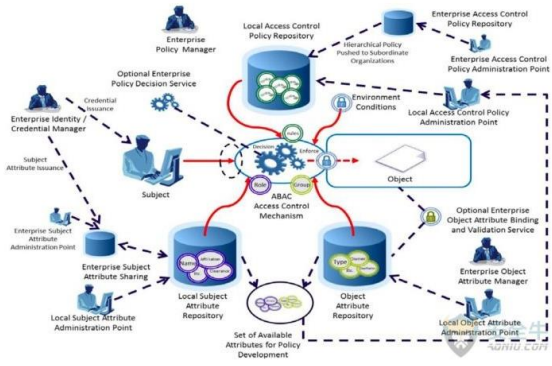
基本访问控制机制比较

访问控制模型	主要优点	主要缺点
DAC	基于授权者的访问控制手段，访问控制灵活	安全可靠性低，授权过于灵活
MAC	基于管理的信息流控制原则，支持多级安全级别安全，具有较高的安全性	授权方式不灵活，安全级别的划分困难
RBAC	角色、集成和约束这些概念与现实世界相吻合，易实现，授权灵活	最小权限约束还不够细化，不具有动态性，不能实现动态的访问控制

基于属性的访问控制：ABAC模型

Attribute-Based Access Control

- 定义表达资源和主体二者属性条件的授权
- 即使用属性定义并实施访问控制，提供上下文相关的细粒度动态访问控制服务。
- 三个关键要素
  - 主体属性，资源属性，环境属性
  - 用户向资源发送请求，授权引擎根据主体（subject）所携带的属性进行判断，确定拒绝或者同意访问资源。
- 管理复杂性



企业ABAC技术使用场景示例

隐藏通道（Covert Channel）

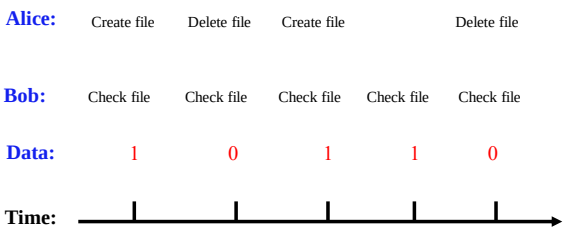
隐藏通道（Covert Channel）问题

- MLS的设计目标：限制通信的合法通道
- 信息流动方式
- 例如，某个资源对象被处于不同安全等级的多个主体所共享
  - 用于传递信息，违反MLS的安全性要求
- 隐藏通道（**Covert channel**）
  - a communication path not intended as such by system’s designers

隐藏通道的例子

- Alice的权限：绝密TOP SECRET，Bob的权限： 机密CONFIDENTIAL，
- 所有用户共享文件空间
  - Bob不能读Alice创建的文件内容，但可以查询文件系统
- Alice 创建文件 FileXYzW，表示给Bob发送信号“1”，删除文件表示 “0”
- Bob每分钟检测该文件
  - If file FileXYzW does not exist, Alice sent 0
  - If file FileXYzW exists, Alice sent 1
- Alice 会向Bob泄露TOP SECRET 信息

隐藏通道的例子



隐藏通道的例子（续）

- 其他可能的隐藏通道?
  - Print queue
  - ACK messages
  - Network traffic, etc.
- 隐藏通道存在的三个条件
  - 共享资源：Sender and receiver have a shared resource
  - 资源属性可观测：Sender able to vary some property of resource that receiver can observe
  - 通信：“Communication” between sender and receiver can be synchronized

隐藏通道

- 隐藏通道无处不在
- 如何清除?
  - Eliminate all shared resources…
  - …and all communication
- 实际上，无法消除所有隐藏通道
  - 破坏系统的可用性
  - DoD 的指导原则: 降低隐蔽信道的容量，例如不超过 1 bit/second
  - 问题：限制是否有效?

隐藏通道

- 例如，一个 100MB **绝密**文件
  - 明文存放在 绝密位置
  - 密文 (encrypted with AES using 256-bit key) 存放在 **公开** 位置
- 假设，隐藏通道容量为 1bps
  - 泄露整个文档花费超过25年
  - 但是，泄露256-bit AES 密钥仅需要不到5分钟

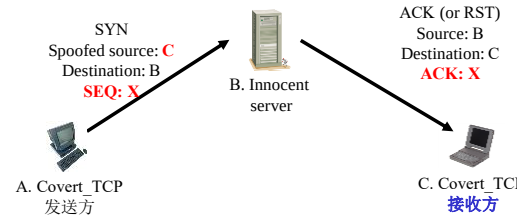
TCP协议：一个实际的例子

TCP报文格式

- ◆隐藏通道
  - ❖利用 “保留reserved” 字段
- ◆使用其他字段隐藏数据
  - ❖序号 Sequence number
  - ❖确认号 ACK number

一个实际的例子（续）

- ◆隐藏数据： TCP sequence numbers
- ◆序号 X 包含隐藏信息



其他隐蔽信道

- ◆分类
  - ❖存储隐蔽信道
  - ❖时间隐蔽信道
    - ➢普通时间隐蔽信道
    - ➢网络时间隐蔽信道
- ◆利用各种网络协议建立隐蔽信道
  - ❖DNS
  - ❖HTTP
  - ❖FTP
  - ❖IPSec

补充：访问控制机制： CAPTCHA

区分计算机和人

图灵测试（Turing Test）

- 1950年，计算机先驱Alan Turing提出著名的图灵测试
- 在测试中，有人向另一个人和计算机提问，发问者无法看到对方
  - 如果计算机能在5分钟内回答由人类测试者提出的一系列问题，且其超过30%的回答让测试者误认为是人类所答，则计算机通过测试，并被认为具有人类智能。
- 人工智能领域的黄金准则
- 进展
  - 在“图灵测试2014”大会上，举办方英国雷丁大学发布新闻稿：“2014图灵测试大会共有5个聊天机器人参与，其中尤金成功地被33%的评委判定为人类。”
    - 尤金：一个13岁的非英语母语的乌克兰男孩

CAPTCHA 验证码

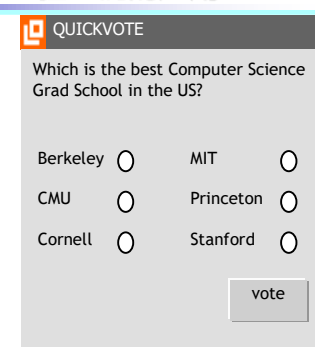
- CAPTCHA 完全自动化的公开的图灵测试
  - 区分计算机和人
  - 可以看成是反向图灵测试
- Completely Automated Public Turing test to tell Computers and Humans Apart
  - 自动化 Automated — 由计算机程序生成测试并评分
  - 公开 Public — 程序和数据公开
  - 图灵测试 Turing test to tell… — 人类能通过测试，而计算机不能

CAPTCHA 悖论?

- 悖论 — 计算机无法通过该测试
  - “…CAPTCHA is a program that can generate and grade tests that it itself cannot pass…”
  - 计算机能够创建测试并进行评分，但自己无法通过测试！
- CAPTCHA 是一种访问控制机制
  - 全自动区分计算机和人类的图灵测试
  - **防止非自然人访问资源对象**
  - 或只有自然人能访问 (i.e., no bots/computers)

CAPTCHA 验证码的应用

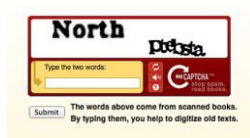
- 2000年，美国卡耐基梅隆大学（CMU）的Luis von Ahn, Manuel Blum, Nicholas Hopper，John Langford提出
  - CAPTCHA的图片由机器生成，控制程序知道正确答案，其目的是为了防止机器自动注册
- 原始动机：在线投票问题
  - 防止自动重复性投票



(**FROM** WWW.SLASHDOT.ORG)

CAPTCHA 验证码的应用

- 免费电子邮件系统
  - 数据收集
  - 蠕虫（Worms），垃圾邮件（Spam）
  - 防止字典攻击（ Dictionary Attacks ）
- 网站防止搜索引擎自动爬取数据
- 采用CAPTCHA的目的
  - 使大部分人容易通过
  - 机器很难或无法通过



CAPTCHAs

- 复杂的CAPTCHA对文字识别，图形图像处理以及人工智能专家来说都是一个很大的挑战
- 目前CAPTCHAs的类型
  - 基于视觉的Visual
  - 基于听觉的Audio
- CAPTCHA测试包租业务
  - 由人去解决CAPTCHA测试问题：利用某些现成的网站的高流量，让那些过路者免费帮忙输入验证码
  - 攻击成本最小化

reCAPTCHA

- 美国卡耐基梅隆大学（CMU）设计
  - OCR（光学自动识别）软件无法识别的文字扫描图作为验证码图片
  - 用户在正确识别出这些文字之后，其答案便会被传回CMU
  - 利用CAPTCHA的原理，借助于人类大脑对难以识别的字符的辨别能力，进行对古旧书籍中难以被OCR识别的字符进行辨别的技术
- 目标
  - 反spam（垃圾邮件），古籍数字化（人工OCR）
- 2009年被google收购
  - 基于云的验证码系统，通过结合程序生成的验证码和较难被OCR识别的图片，来帮助Google数字化一些书籍，报纸和街景里的门牌号等

reCAPTCHA

◆工作原理

1、扫描书籍

2、提取OCR无法识别的单词

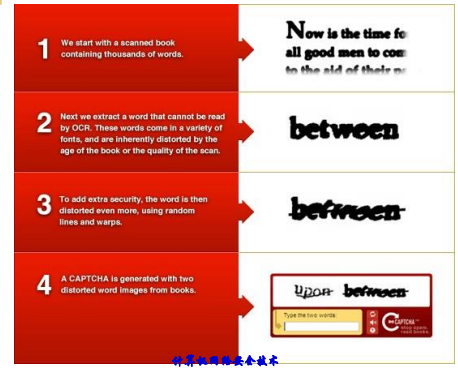
3、进一步扭曲并加入随机横线来增强安全性

4、使用两个单词生成验证码来让用户识别

◆方法

❖“两个验证码里面有一个是被人审核过的，而另一个是计算机读不出来的。当你把那个正确的输对以后，系统就会默认另外一个也是对的。”

reCAPTCHA



reCAPTCHA 里显示 Google 街景的图片

reCAPTCHA–noCAPTCHA

◆2014年12月，谷歌发表文章

❖*Are you a robot? Introducing “No CAPTCHA reCAPTCHA”*

❖研究表明现在的人工智能技术已经能够解决99.8%的验证码，因此扭曲文本的验证方式可能不是一个可靠的方法。

◆“No CAPTCHA reCAPTCHA”

❖用户只需要简单的勾选就可以确认你是真实用户而非恶意机器人

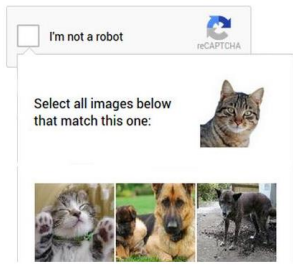
❖系统会记录并且预检每个用户的行为，并过滤掉任何被容易识别为人类的用户。

❖“风险分析引擎”，机器学习技术

谷歌发布改良版CAPTCHA 证明你是真人更容易

腾讯科技 [微博] 2014年12月04日05:14 分享

[摘要]CAPTCHA让你从一组图片中挑选出正确的图片，选对了，就能证明“我不是机器人”。



补充：OAuth

单点登录SSO技术：OAuth

- OAuth（开放授权）
  - 是Open Authorization的缩写，为用户资源的授权提供了一个安全的、开放而又简易的标准。
  - 在用户授权的前提下允许第三方网站访问用户在服务商里存储信息。
  - 这种授权无需将用户提供用户名和密码提供给该第三方网站。
  - 提供一个令牌给第三方网站，一个令牌对应一个特定的第三方网站，同时该令牌只能在特定的时间内访问特定的资源。
- 例子
  - 有一个"云冲印"的网站，可以将用户储存在Google的照片，冲印出来。用户为了使用该服务，必须让"云冲印"读取自己储存在Google上的照片。
  - 问题： “云冲印”怎样获得用户的授权?

定义

（1） Third-party application：第三方应用程序，又称“客户端”（client），如"云冲印"。

（2）HTTP service：HTTP服务提供商，简称“服务提供商”，如Google。

（3）Resource Owner：资源拥有者，又称"用户"（user）

（4）User Agent：用户代理， 即指浏览器。

（5）Authorization server：认证服务器，即服务提供商专门用来处理认证的服务器。

（6）Resource server：资源服务器，即服务提供商存放用户生成的资源的服务器。它与认证服务器，可以是同一台服务器，也可以是不同的服务器。

实例

- 终端用户（资源拥有者）可以许可一个打印服务（客户端）访问她存储在图片分享网站（资源服务器）上的受保护图片，而无需与打印服务分享自己的用户名和密码。
- 她直接与图片分享网站信任的服务器（验证服务器）进行身份验证和授权，该服务器颁发给打印服务具体委托凭据（访问令牌）。

OAuth2.0授权基本流程

（A）用户打开客户端以后，客户端要求用户给予授权。

（B）用户同意给予客户端授权。

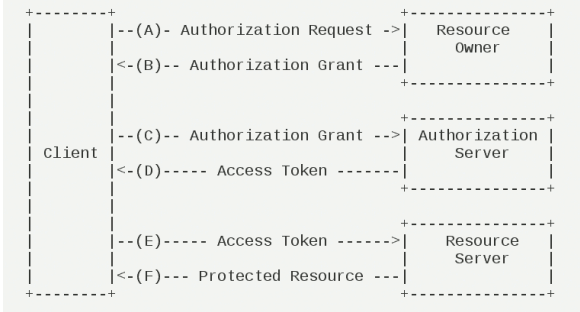
（C）客户端使用上一步获得的授权，向认证服务器申请令牌。

（D）认证服务器对客户端进行认证以后，确认无误，同意发放令牌。

（E）客户端使用令牌，向资源服务器申请获取资源。

（F）资源服务器确认令牌无误，同意向客户端开放资源

OAuth 2.0的运行流程（RFC 6749）



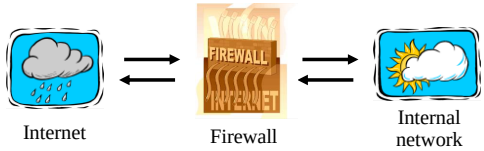
OAuth 2.0授权方式

- OAuth 2.0定义了四种授权方式
  - 授权码模式（authorization code）
  - 简化模式（implicit）
  - 密码模式（resource owner password credentials）
  - 客户端模式（client credentials）

三、网络安全基础设施

防火墙 Firewalls

防火墙 Firewalls



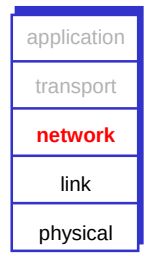
- 网络的访问控制机制
- 防火墙确定过滤或转发出入内网的报文

有关术语

- 防火墙类型
  - 包过滤防火墙Packet filter
    - 工作在网络层
  - 基于状态检测的包过滤防火墙 Stateful packet filter
    - 工作在传输层
  - 应用层代理 Application proxy
    - 应用层
  - 链路级网关 Circuit Level Gateway
    - 工作在传输层

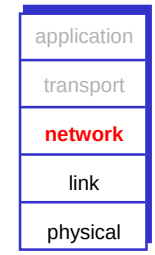
包过滤防火墙

- 工作在网络层
- 过滤条件
  - 源IP地址 Source IP address
  - 目的IP地址 Destination IP address
  - 源端口 Source Port
  - 目的端口 Destination Port
  - 标志位 Flag bits (SYN, ACK, etc.)
  - 流出Egress/进入 ingress



包过滤防火墙

- 优点?
  - 速度快 Speed
- 缺点?
  - 缺乏状态的概念
  - 无法看见TCP连接
  - 无法分析应用数据（深度包检测DPI）



Linux中的包过滤防火墙Netfilter

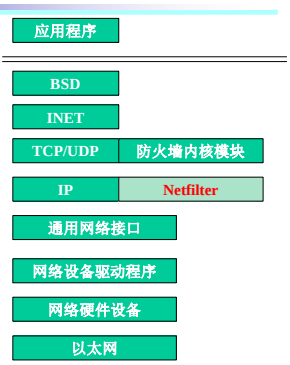
（补充）

Linux中的包过滤防火墙Netfilter

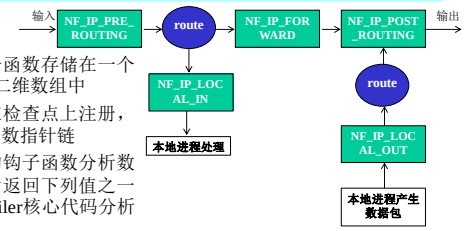
- Linux中的包过滤防火墙Netfilter
  - 包含在Linux 2.4以后的内核中，可以实现防火墙、NAT（网络地址翻译）和数据包的分割等功能。
  - netfilter工作在内核内部，而iptables则是让用户定义规则集的表结构
  - Netfilter/iptables从ipchains和ipwadfm（IP防火墙管理）演化而来，功能增强
- 功能
  - 包过滤：filter表不会对数据报进行修改，而只对数据报进行过滤
  - NAT：NAT表监听三个netfilter钩子函数：NF_IP_PRE_ROUTING、NF_IP_POST_ROUTING及NF_IP_LOCAL_OUT
  - 数据报处理：mangle表在NF_IP_PRE_ROUTING和NF_IP_LOCAL_OUT钩子中进行注册，对数据报的修改。

Netfilter在内核中的位置

- IP和防火墙内核模块之间
- IP和Netfilter结构相对独立
- 为每种网络协议（IPv4、IPv6、IPX等）定义一套**钩子函数（hook）**
- 数据报流经协议栈的几个检查点时被调用
- 对报文进行处理（修改、丢弃或传送给用户进程）
- 使用IPTables组件在用户空间对Netfilter进行设置，定制自己的防火墙



Netfilter检查点



- 定义的钩子函数存储在一个list_head的二维数组中
- 钩子函数在检查点上注册，形成一条函数指针链
- 每个注册的钩子函数分析数据报结束后返回下列值之一，告知Netfiler核心代码分析结果
  - NF_ACCEPT：允许数据报文通过，进入下一步处理
  - NF_DROP：丢弃该报文
  - NF_STOLEN：由钩子函数处理该数据报，不再继续传送
  - NF_QUEUE：将数据报加入队列，交由用户程序处理
  - NF_REPEAT：再次调用该钩子函数

IPTables

◆**Netfilter/IPTables**

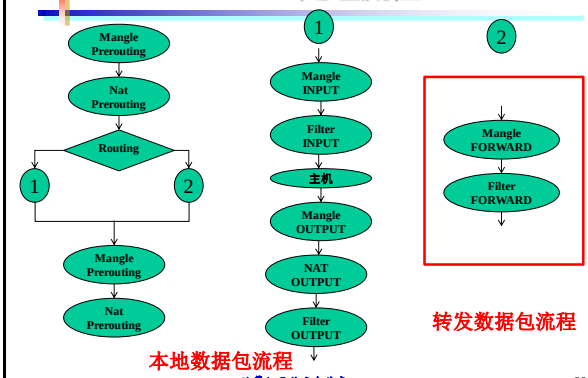
- ❖iptables组件是用户空间（userspace）的工具，插入、修改和删除信息包过滤表中的规则。
- ❖Netfilter的钩子函数
- ❖指导这些钩子函数如何工作的一系列规则

◆**钩子函数通过访问表中的规则判断应该返回什么值给Netfilter模块**

◆**内建表**

- ❖Mangle，Nat
- ❖Filter ： NF_IP_LOCAL_IN 、 NF_IP_FORWARD 、 NF_IP_LOCAL_OUT三处组册了钩子函数
- ❖注入模块的方式，调用Netfilter的接口函数创建新的表

Netfilter处理流程



本地数据包流程

转发数据包流程

动态包过滤例子

◆Linux中的Netfilter

◆**网络连接跟踪信息/proc/net/ip_conntrack**

❖Tcp 6 117 SYN-SENT src=192.168.1.6 dst=35.115.106.88

协议 | 协议号 | 生命期 | 状态

sport=32775 dport=22**[UNREPLIED]** src=35.115.106.88

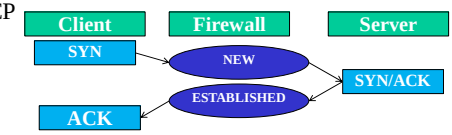
未收到任何响应

dst=211.154.172.2 sport=22 dport=32775 use=2

NAT

动态包过滤例子

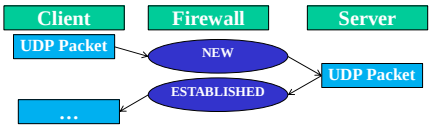
◆TCP



- ❖Tcp 6 117 **SYN-SENT** src=192.168.1.5 dst=192.168.1.35 sport=1031 dport=23**[UNREPLIED]** src=192.168.1.35 dst=192.168.1.5 sport=23 dport=1031 use=2
- ❖Tcp 6 57 **SYN-RECV** src=192.168.1.5 dst=192.168.1.35 sport=1031 dport=23 src=192.168.1.35 dst=192.168.1.5 sport=23 dport=1031 use=1
- ❖Tcp 6 631999 **ESTABLISHED** src=192.168.1.5 dst=192.168.1.35 sport=1031 dport=23 src=192.168.1.35 dst=192.168.1.5 sport=23 dport=1031 use=1

动态包过滤例子

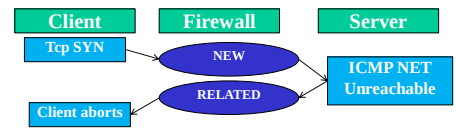
◆UDP



- Udp 17 20 src=192.168.1.2 dst=192.168.1.5 sport=137 dport=1025**[UNREPLIED]** src=192.168.1.5 dst=192.168.1.2 sport=1025 dport=137 use=1
- Udp 17 170 src=192.168.1.2 dst=192.168.1.5 sport=137 dport=1025 src=192.168.1.5 dst=192.168.1.2 sport=1025 dport=137 use=1
- Udp 17 175 src=192.168.1.5 dst=192.168.1.2 sport=137 dport=1025**[ASSURED]** src=192.168.1.5 dst=192.168.1.2 sport=1025 dport=137 use=1

动态包过滤例子

◆TCP和ICMP的关联



◆其他协议

- 类似于UDP，第一个包认为NEW，其后的应答包都是ESTABLISHED

作业

◆访问控制列表和访问能力列表的区别是什么?

◆RBAC的基本模型是什么?

◆举例说明混淆代理人攻击

◆举例说明隐蔽信道攻击

◆ 下次课（4月10日）大作业开题讨论，请及时在课程中心提交第一次大作业